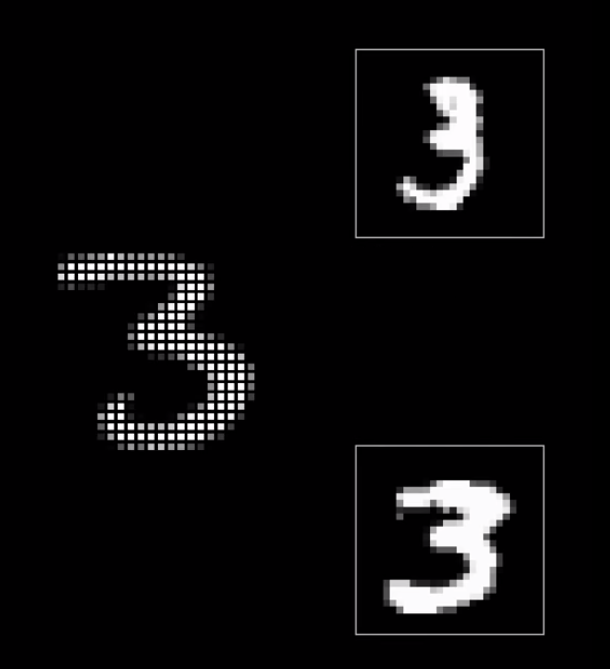
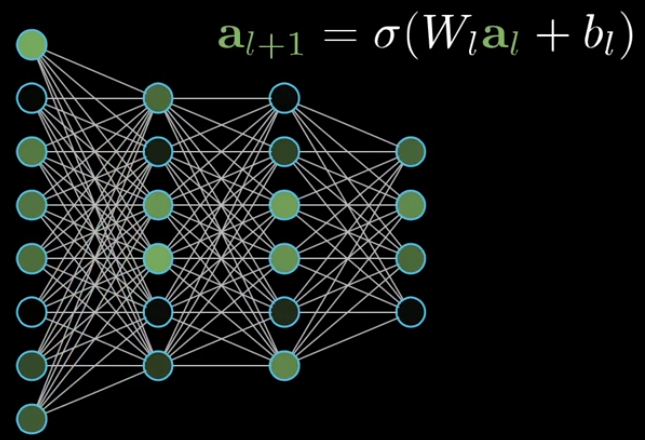
# Neural networks

Imágenes de números borrosas pueden ser fácilmente reconocidas por el ojo humano, incluso si tienen mucha diferencia entre sí. Las partículas de luz que recibimos en nuestros ojos son muy diferentes entre cada ejemplo.



Para el ojo humano es una tarea extremadamente sencilla, pero si tuviéramos que construir un programa que diferenciara esto, sería algo extremadamente difícil.

0

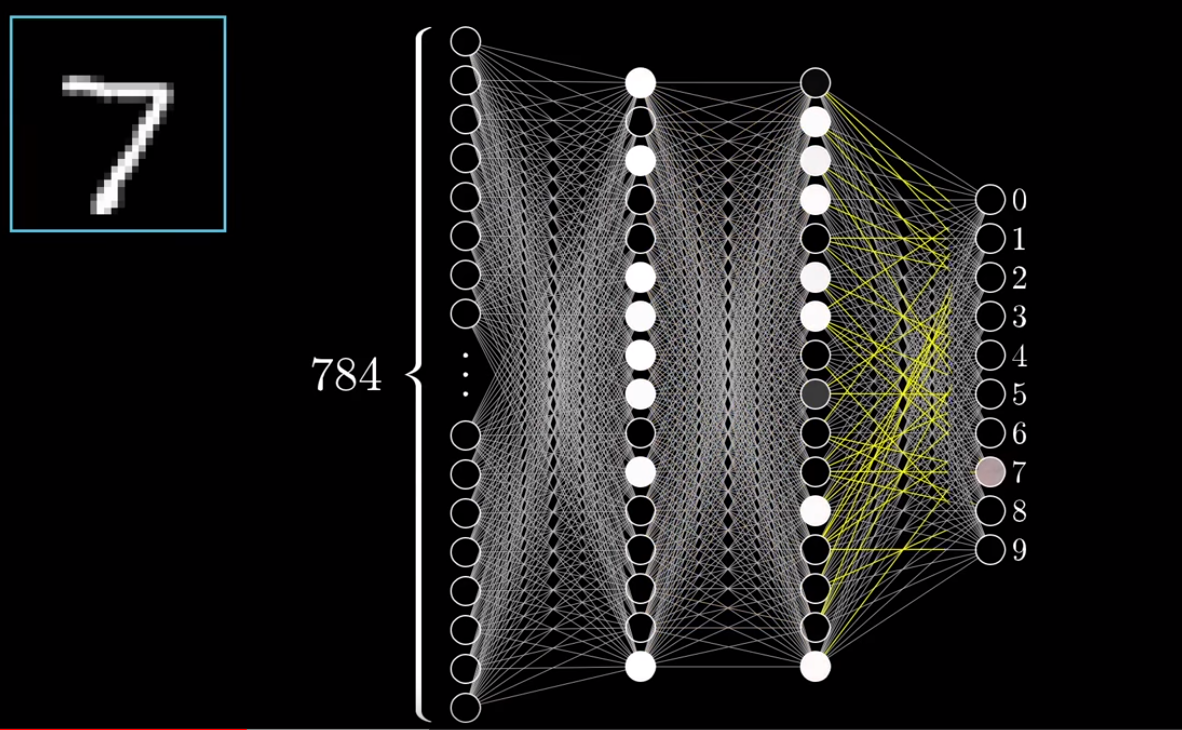
Neuron => a thing that holds a number, specificaly between 0 and 1

Si tenemos una imagen de 28x28 = 784

Vamos a decir que cada una de las 784 neuronas tiene un valor de 0 para pixeles negros y 1 para los blancos. El valor de cada pixel es llamado “ Activation “

La primera fila de nuestra red tendrá 784 neuronas, y la última solo 10. Idealmente solo se tendría que activar una sola neurona al final con un score, que indica cuanto piensa la red neuronal que el número analizado es el de la neurona activada. Por ejemplo, si la neurona 7 tiene un 0.9, el programa dirá que el número analizado tiene un 90% de coincidencia con el 7.

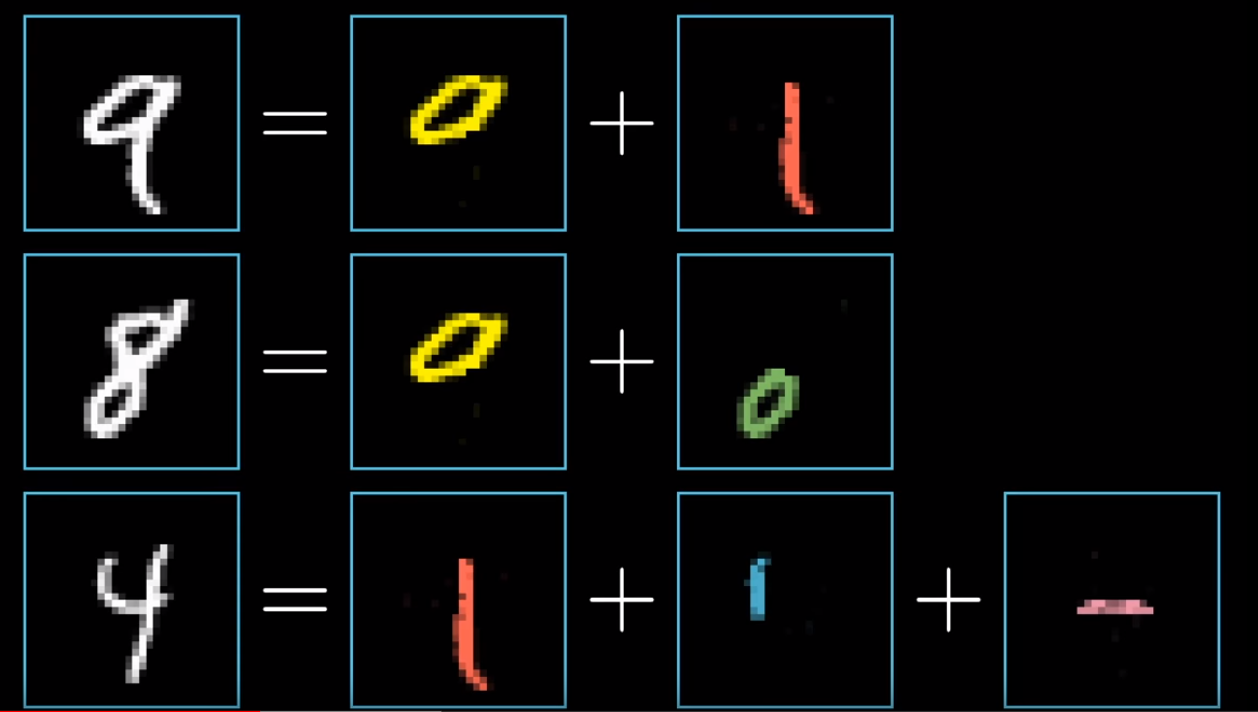
Las capas del medio son llamadas Hidden Layers. Las activaciones de la layer anterior condicionan la siguiente. Esto es una analogía a nuestras neuronas, ya que el impulso eléctrico de una dispara la activación de las siguientes. Podemos pensar la red neuronal como una función, que toma 784 parametros de entrada y devuelve 10.



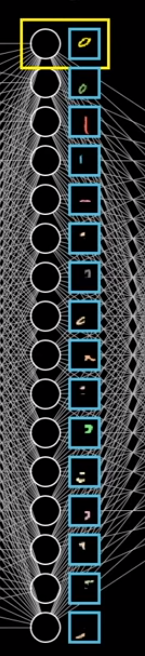
¿Cómo una columna influencia la siguiente?

¿Cómo se entrenan?

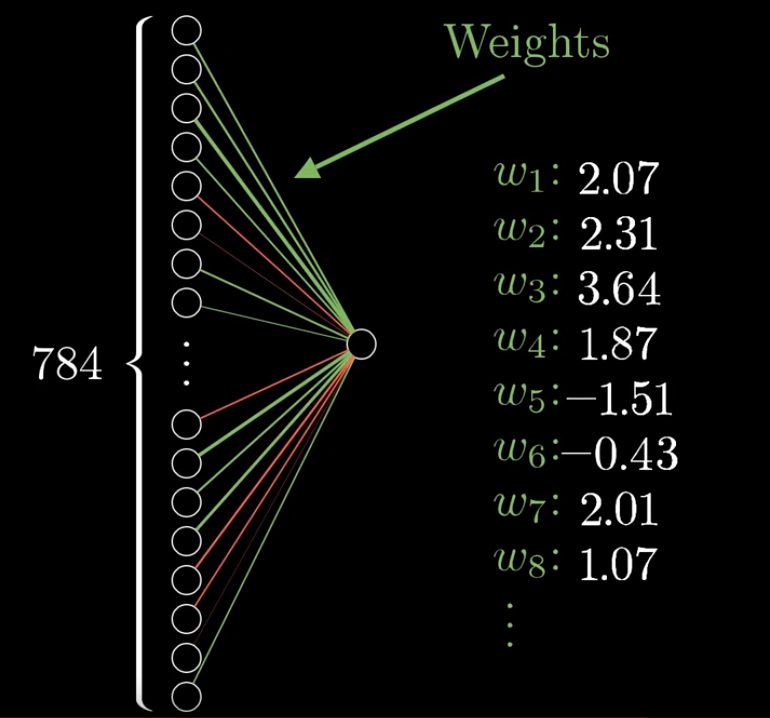
# ¿Por qué usamos layers?

Cuando reconocemos o damos sentidos a lo que vemos, nuestro cerebro combina cosas. Por ejemplo, un nueve podría tener un circulo arriba y una línea abajo. Un ocho es una combinación de dos círculos, etc. 

Entonces, cuando le damos un número a la red neuronal, diferentes neuronas se activaran dependiendo de los componentes de la imagen.



Reconocer un circulo puede traernos otros problemas. Una forma de abordarlo sería reconocer los costados que lo conforman.

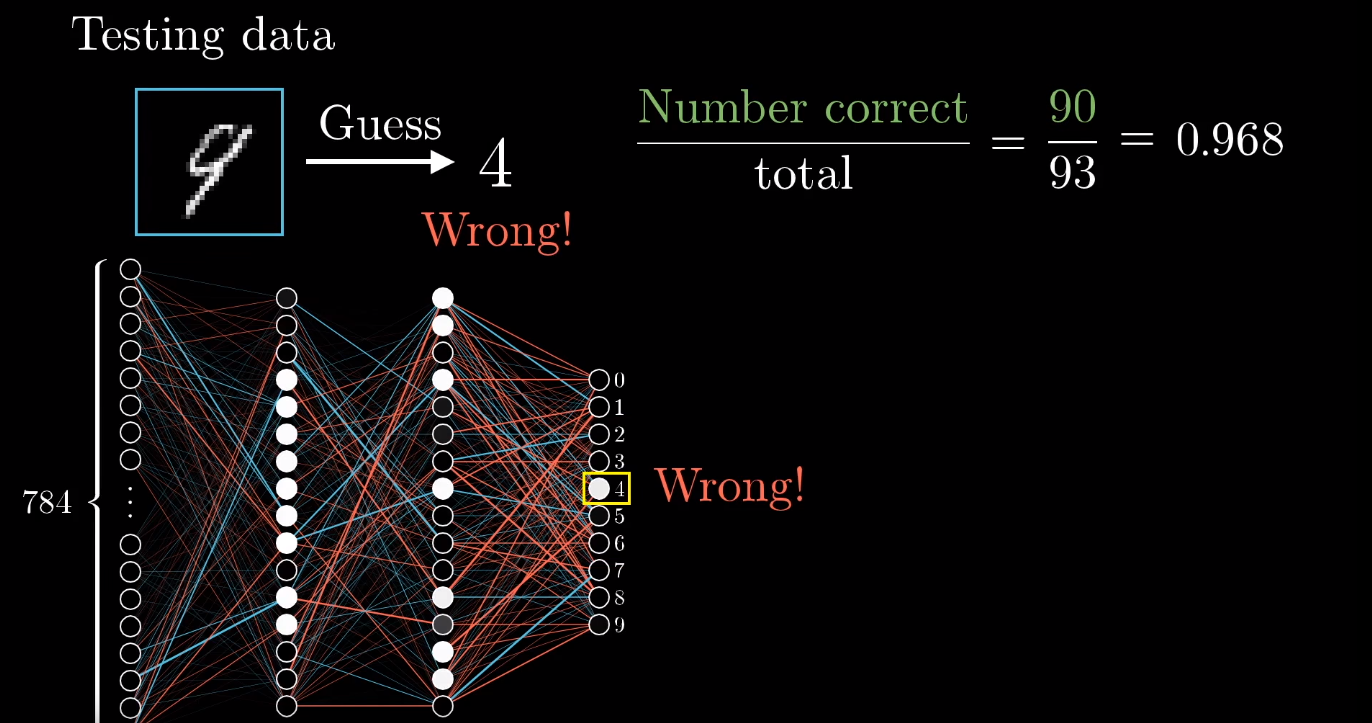


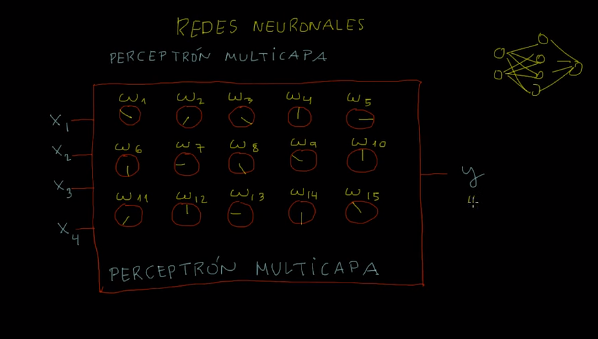
Asignación de peso. Son simplementoe números. Entonces agarramos todas las neuronas activadas y las combinamos con supeso. Gracias a esto, podemos hacer que la primera layer analice bordes y la segunda patrones.

Linear algebra, chapter 3.

La función de coste es una capa de complejidad encima de todo esto.

Toma un input y devuelve una medida de Lousiness (cuan errada está la máquina), que sería el coste.Los parámetros son muchos ejemplos de entrenamiento.





Recompilamos cosas que ya sabemos

Parejas que se han divorciado han tenido:

X1 = 10, x2=20, x=4, etc

Al principio da un valor absurdo, porque la red no ha aprendido nada. Etapa de aprendizaje: para cada conjunto de valores, cada vez que me daba una cosa que distaba mucho de lo que tenía que hacer, lo que se hace es calcular el error. Si tenía que dar 1, y da 0.3, el error es de 0.7. épocas.

Si me llega un matrimonio nuevo, que no está acá, vamos a poder predecir el resultado

1. Aprendizaje, tengo otros 50 casos que no usé para entrenar a la red
2. Validación. Contrastamos los otros casos y los introducimos en la red, comparándolas con lo que tendría que haber dado. Si el resultado es muy cercano, ha funcionado.

# Descenso del gradiente

F(w) = (x-1)ˆ2 Objetivo: calcular w\* (valor de omega que minimiza f(w)

Approach clásico: si la función cumple todos los requisitos para poder derivarse, calculamos la derivada y ese valor será el que maximiza o minimiza.

En las redes neuronales, este método no es efectivo, porque tendremos una cantidad x de omegas. Utilizamos el descenso del gradiente.

Partimos de un cierto omega, y apuntamos lo que vale F(w)

|  |  |
| --- | --- |
| W | F(W) |
| 2 | 1 |
| 1.9 | 0.9 |
|  |  |
|  |  |
|  |  |

Luego del primero, qué debería probar, 2,1 o 1,9? Cuál de los dos se acerca al minimo del F. Con 1.9 obtenemos un menor resultado. La idea es preguntrase para dónde variar la omega para ir acercarndonos al mínimo. Cuando tengmaos que modificar la omega, **hagamoslo siempre en contra de la derivada.** La derivada nos informa de la recta pendiente. Si la derviada da positiva, quiere decir que si aumentamos omega, la función aumenta, y viceversa. Entonces, ir en contra de la derivada hace que la función F disminuya.

* W(0)
* W(1) = W(0) – F(W(0))

Ej:

W(0) = 2

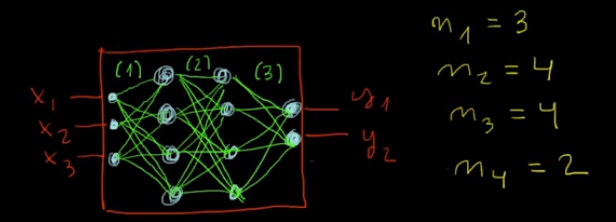
W1 = 2-2(2-1) = 2-2 = 0;

W(2) = 0-2(0-1) = 2;

Este proceso va y vuelve. La estrategia es generalizar la regla y decir que la Omega en el paso siguiente, que sea la omega anterior – la derivada, pero multiplicaremos la derivada por un número alfa (llamado razón de aprendizaje) el valor depende del problema.

Derivada parcial: Cuanto varía el error si se modifica una sola omega.

N es el número de neuronas de la capa. Siendo n la capa



|  |  |
| --- | --- |
| N1 | 3 |
| N2 | 4 |
| N3 | 4 |
| N4 | 2 |

Los números (1) representan las conexiones sinápticas. El primer número representa la primer neurona, y el segundo la de la siguiente capa.

Ej (1) W11, (1)W12 ….. (1) W IJ

O = 1, …. m1

J = 1, …. m2